Adaptively Indistinguishable Garbled Circuits

Zahra Jafargholi* Alessandra Scafuro † Daniel Wichs‡

Abstract

A garbling scheme is used to garble a circuit C and an input x in a way that reveals the output $C(x)$ but hides everything else. An adaptively secure scheme allows the adversary to specify the input x after seeing the garbled circuit. Applebaum et al. (CRYPTO '13) showed that in any garbling scheme with adaptive simulation-based security, the size of the garbled input must exceed the output size of the circuit. Here we show how to circumvent this lower bound and achieve significantly better efficiency under the minimal assumption that one-way functions exist by relaxing the security notion from simulation-based to indistinguishability-based.

We rely on the recent work of Hemenway et al. (CRYPTO '16) which constructed an adaptive simulation-based garbling scheme under one-way functions. The size of the garbled input in their scheme is as large as the *output size* of the circuit plus a certain *pebble complexity* of the circuit, where the latter is (e.g.,) bounded by the space complexity of the computation. By building on top of their construction and adapting their proof technique, we show how to remove the output size dependence in their result when considering indistinguishability-based security.

As an application of the above result, we get a symmetric-key functional encryption based on one-way functions, with indistinguishability-based security where the adversary can obtain an unbounded number of function secret keys and then adaptively a single challenge ciphertext. The size of the ciphertext only depends on the maximal pebble complexity of each of the functions but not on the number of functions or their circuit size.

*Aarhus University, Denmark. zahra@cs.au.dk
†North Carolina State University, USA. ascafur@ncsu.edu
‡Northeastern University, USA. wichs@ccs.neu.edu

1 Introduction

Garbled Circuits. A *garbling scheme* [Yao82, Yao86] can be used to garble a circuit C and an input x to derive a garbled circuit $\widetilde{C}$ and a garbled input $\tilde{x}$. It's possible to evaluate $\widetilde{C}$ on $\tilde{x}$ and get the correct output $C(x)$. However, the garbled values $\widetilde{C}, \tilde{x}$ should not reveal anything else beyond this. In many applications, the garbled circuit $\widetilde{C}$ can be computed in an *off-line* pre-processing phase before the input is known and therefore we are not overly concerned with the efficiency of this procedure. On the other hand, once the input x becomes available in the *on-line* phase, creating the garbled input $\tilde{x}$ should be extremely efficient. Therefore, the main efficiency measure that we consider here is the *on-line complexity* of a garbling scheme, which is the time it takes to garble an input x, and hence also a bound on the size of $\tilde{x}$.

Security of Garbled Circuits. There are several natural notions of garbled circuit security that one can consider.

Firstly, we can consider either *selective* or *adaptive* security. For selective security, we consider a scenario where the adversary chooses the circuit C and the input x first and only then gets the garbled versions $\widetilde{C}, \tilde{x}$. For adaptive security, we consider a scenario where the adversary first gets the garbled circuit $\widetilde{C}$ and can then adaptively chooses the input x to be garbled. Adaptive security is the natural notion in the on-line/off-line setting where we envision the garbled circuit to be created first in an earlier stage before the input is selected.

Secondly, we can consider either *simulation-based* or *indistinguishability-based* definitions of security. In the simulation-based setting, we require that the garbled circuit and the garbled input can be simulated given only the output of the computation and the topology of the circuit. In the indistinguishability-based setting, we require that the adversary cannot distinguish between a garbling of C_0, x_0 or C_1, x_1 as long as $C_0(x_0) = C_1(x_1)$ and C_0, C_1 have the same topology.

Prior Work. Yao's construction of garbled circuits under one-way functions already achieves essentially optimal on-line complexity, where the time to garble an input x and the size of $\tilde{x}$ are only linear in the input size $|x|$, independent of the circuit size.[1] However, it was only shown to satisfy *selective* simulation-based security [LP09].

Recently, the work of Hemenway et al. [HJO+16] showed how to modify Yao's construction and get *adaptive* simulation-based security under one-way functions. The on-line complexity of their scheme depends linearly on a certain "*pebble complexity*" t of the circuit, its *input size* n and *output size* m. Furthermore, they showed that the pebble complexity t is upper bounded by the circuit width which is in turn bounded by the space complexity of the computation. The work of [JW16] (see also [JKK+17]) shows that even Yao's original garbled circuit construction already achieves adaptive simulation-based security via reduction with a 2^t security loss as long as the mapping between output labels and the bits they represent is only given in the garbled input.

In both of the above works, the online complexity is always at least as large as the output size m. The work of Applebaum et al. [AIKW13] (see also [HW15]) gives a lower bound showing that this is inherent for adaptive simulation-secure garbled circuits.

Our Results. In this work, we show how to construct adaptively secure garbling schemes based on one-way functions, where the on-line complexity of our scheme can be smaller than the output size of the circuit. This necessarily requires us to give up on simulation-based security and instead we achieve indistinguishability-based security. In more detail, we propose a new garbling scheme which builds on top of the ideas of [HJO+16] but essentially removes the output size dependence in their construction, making the on-line complexity only linear in the pebble complexity t and the input size n, but independent of the output size m.

As an application of the above result, we consider the scenario where we garble a circuit C which consists of many disjoint boolean sub-circuits $C_1, \ldots, C_\ell$ which all take the same input x but do not share any other wires/gates except for the input wires. In that case, although the output size of C is ℓ (which we think of as large) the pebble complexity of C is just $t = \max\{t_i\}$ where t_i denote the pebble complexities of the

[1] More precisely, in Yao's garbled circuits, the garbled input is of size $|x| \cdot \mathsf{poly}(\lambda)$ where λ is the security parameter. The work of Applebaum et al. [AIKW13] shows how to reduce this to $|x| + \mathsf{poly}(\lambda)$ assuming stronger assumptions such as DDH, RSA or LWE.

individual circuits C_i, and therefore is independent of the number of circuits ℓ. We can also think of the above as allowing us to construct an adaptively indistinguishable private-key functional encryption (FE) scheme by thinking of the garbled versions of the circuits C_i as function secret keys and the garbled input as a ciphertext. The size of the ciphertext is linear in the size of the input x and the maximal pebble complexity of the individual functions, which we can bound by their space complexity, but is independent of the number of function secret keys ℓ or even their circuit size.

Finally it bears mentioning that an adaptively indistinguishable scheme is also adaptively secure under the simulation-based security definition for any efficiently invertible function.[2] Therefore for this class of functions our construction provides a *simulation-based adaptively secure* garbling scheme with online complexity independent of the output size.

1.1 Our Techniques

Before we can explain our techniques, we first review Yao's garbled circuit construction, the issue with adaptive security and the technique of [HJO+16]. The discussion below is adapted from [HJO+16].

Yao's Scheme. First, let's start by recalling Yao's garbled circuits. For each wire w in the circuit, we pick two keys k_w^0, k_w^1 for a symmetric-key encryption scheme. For each gate in the circuit computing a function $g : \{0,1\}^2 \to \{0,1\}$ and having input wires a, b and output wire c we create a *garbled gate* consisting of 4 randomly ordered ciphertexts created as:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{g(0,0)})) \qquad & c_{1,0} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{g(1,0)})), \\ c_{0,1} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{g(0,1)})) \qquad & c_{1,1} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{g(1,1)})) \end{aligned} \tag{1}$$

where $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure encryption scheme. The garbled circuit $\widetilde{C}$ consists of all of the gabled gates, along with an *output map*

$$\{k_w^0 \to 0, k_w^1 \to 1\}$$

which maps the keys to the bits they represent for each output wire w. To garble an n-bit value $x = x_1 x_2 \cdots x_n$, the garbled input $\tilde{x}$ consists of the keys $k_{w_i}^{x_i}$ for the n input wires w_i.

To evaluate the garbled circuit on the garbled input, it's possible to decrypt exactly one ciphertext in each garbled gate and get the key $k_w^{v(w)}$ corresponding to the bit $v(w)$ going over the wire w during the computation $C(x)$. Once the keys for the output wires are computed, it's possible to recover the actual output bits by looking them up in the output map.

To prove the selective simulation-based security of Yao's scheme, we have a simulator that gets the output $y = y_1 y_2 \cdots y_m = C(x)$ and must produce $\widetilde{C}$, $\tilde{x}$. The simulator picks random keys k_1^0, k_w^1 for each wire w just like the real scheme, but it creates the garbled gates as follows:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^0)) \qquad & c_{1,0} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^0)), \\ c_{0,1} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^0)) \qquad & c_{1,1} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^0)) \end{aligned} \tag{2}$$

where all four ciphertext encrypt the same key k_c^0. It then sets the output map as $\{k_w^0 \to y_w, k_w^1 \to 1 - y_w\}$ by "programming it" so that the key k_w^0 corresponds to the correct output bit y_w for each output wire w. This defines the simulated garbled circuit $\widetilde{C}$. To create the simulated garbled input $\tilde{x}$ the simulator simply gives out the keys k_w^0 for each input wire w. Note that, when evaluating the simulated garbled circuit on the simulated garbled input, the adversary only sees the keys k_w^0 for every wire w.

Proof of Security and Issues with Adaptivity. There are two main issues with proving adaptive security of Yao's construction.

The first issue is that, in the simulation-based security setting, the simulator now cannot "program" the output map since it is given as part of the garbled circuit before the output $y_1, \ldots, y_m$ is defined. This can be fixed by modifying the construction and moving the output map from the garbled circuit to the garbled

[2] More generally, any function f for which, given any image element y it is possible to efficiently find a canonical pre-image x.

input, at the cost of raising the on-line complexity to depend on the output size. In the simulation-based setting we know this to be inherent, but one could hope to avoid this in the indistinguishability-based setting.

The second and more serious issue is the sequence of hybrids used to prove security. At a high level, the selective proof proceeds via a series of carefully defined hybrid games that switch the distribution of one garbled gate at a time, starting with the input level and proceeding up the circuit level by level. In addition to the two modes of creating garbled gates defined above, we also define an additional mode where the garbled gate is set to:

$$\begin{aligned} c_{0,0} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})) \qquad & c_{1,0} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)})), \\ c_{0,1} &= \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) & c_{1,1} &= \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)})) \end{aligned} \tag{3}$$

where $v(c)$ is the correct *value* of the bit going over the wire c during the computation of $C(x)$. Let us give names to the three modes for creating garbled gates that we defined above: (1) is called RealGate mode, (2) is called SimGate mode, and (3) is called InputDepSimGate mode, since the way that it is defined depends adaptively on the choice of the input x. The proof of selective security of Yao's garbled circuits proceeds in a sequence of hybrids where the way we garble a gate goes from RealGate mode to InputDepSimGate mode to SimGate mode in some carefully chosen order. The problem with adapting this technique to the adaptive setting is that the InputDepSimGate mode is not (even syntactically) well defined; in this mode the way that we garble the gate depends on the value that the output wire takes on during the computation $C(x)$ but in the adaptive setting the input x is not yet defined when we create the garbled circuit.

The Technique of [HJO+16]. Essentially, the work of [HJO+16] proves adaptive security by leveraging two ideas.

Firstly, they encrypt the entire Yao garbled circuit under an additional layer of encryption using a special "somewhere equivocal encryption scheme", and give the decryption key as part of the garbled input. Such a scheme can be used to create a simulated ciphertext given only some but not all of the plaintext blocks (think of the unknown blocks as "holes") and later create a secret key that decrypts all the known blocks correctly but "plugs the holes" with arbitrarily specified values. The size of the secret key only depends on the number of holes and not the entire size of the plaintext. By leveraging this type of encryption, they can define hybrid games where some of the gates are in InputDepSimGate mode (which is not well defined when the circuit is created) by putting "holes" in place of all such gates when creating the garbled circuit and then coming up with a decryption key that opens the holes to the correct value when creating the garbled input (at which point InputDepSimGate is well defined).

Secondly, the above idea requires the number of holes (and therefore the size of the garbled input) to scale with the number of gates in InputDepSimGate mode in any hybrid. Therefore, to get a non-trivial result, we need a sequence of hybrids that minimizes the number of gates in InputDepSimGate mode at any point in time. Recall that we start with all gates in RealGate mode and want to end with all gates in SimGate mode. We are allowed to make the following changes:

- We can change a gate from RealGate to InputDepSimGate (and back) as long as its predecessors are in InputDepSimGate mode (or it is at the input level). This is because, in this case, only one of the keys for each input wire appears in the game.
- We can change a gate from InputDepSimGate to SimGate (and back) as long as all of its successors are in SimGate mode (or it is at the output level). This is because the two keys associated with the output wire are used interchangeably in the game.

The work of [HJO+16] connects the above with a *pebbling game* over the circuit, where the goal is to change all the gates from RealGate to SimGate subject to the above rules while minimizing the number of gates in InputDepSimGate mode at any point in time: this latter number is defined to be the *pebble complexity* of the circuit. For example, they show that the pebble complexity of a circuit is bounded by its *width* which in turn corresponds to the space complexity of the computation. The size of the garbled input in their scheme is the maximum of the pebble complexity of the circuit and the input/output size.

Our Construction and Proof Technique. One could hope to get rid of output dependence in the construction of [HJO+16] by simply sending the output map (the mapping between the keys of the output wires and the bits they represent) with the garbled circuit rather than with the garbled input. Although we know that such a construction cannot achieve adaptive simulation security, one could conjecture it to achieve adaptive indistinguishability security. Unfortunately, we do not know how to prove such a construction secure. Essentially, the issue is that the only reason we can change output gates from $\mathsf{InputDepSimGate}$ to $\mathsf{SimGate}$ in the proof of [HJO+16] is that we can "program" the output map after the actual output of the computation is known; if the output map is sent with the garbled circuit this is no longer possible. Instead, we come up with a modified construction which we are able to prove secure.

Our new garbling construction leverages that of [HJO+16] and proceeds as follows. To garble a circuit C we use the scheme of [HJO+16] and garble two copies of C completely independently: we call the resulting garbled circuits $\mathsf{C_L}, \mathsf{C_R}$. These are just Yao garbled circuits (without an output map) encrypted under an additional layer of somewhere equivocal encryption. We choose one of the two garbled circuits at random to be the "active" one: $\mathsf{active} \leftarrow \{L, R\}$. Then we merge the two garbled circuits by creating a layer of garbled "selection gates" (s-gates): for each output bit $i \in [m]$ we create an s-gate that takes the i'th output wire from both garbled circuits, and outputs the value on the wire coming from the active circuit (the output of the garbled s-gate is a bit in the clear rather than a wire key). The garbled circuit consists of $\widetilde{C} = (\mathsf{C_L}, \mathsf{C_R}, \widetilde{\mathsf{sgate}})$. To garbled an input x we use the scheme of [HJO+16] to garble two copies of it for the left and right garbled circuit. The evaluation procedure does the natural thing by evaluating both $\mathsf{C_L}, \mathsf{C_R}$ respectively, and using the output wire keys on the garbled s-gates to recover the output bits in the clear. Ideas similar to the use of two circuits along with a selection layer have appeared in prior works, e.g., [PST14].

To prove security, we consider an adversary that chooses C_0, C_1, gets a garbled version of C_b, then adaptively chooses x_0, x_1 such that $C_0(x_0) = C_1(x_1)$, and gets a garbled version of x_b. We want to show that the adversary cannot distinguish between $b = 0$ and $b = 1$. We show security via the following sequence of hybrids.

1. We start with the security game where the challenge bit is $b = 0$. In this case, both $\mathsf{C_L}, \mathsf{C_R}$ garble C_0 and both garbled inputs correspond to x_0. Let $\mathsf{active} \in \{L, R\}$ be the identity of the active circuit. We use the notation $\mathsf{C_{active}}, \mathsf{C_{passive}}$ to denote the active and passive garbled circuits respectively.

2. We change the passive garbled circuit $\mathsf{C_{passive}}$ and the garbled input for it to be simulated. This change essentially follows the proof of [HJO+16]. In particular, we rely on the fact that the keys associated with the bits 0 and 1 for the output wires of $\mathsf{C_{passive}}$ are used symmetrically by the s-gates (since the s-gates are ignoring the output of the passive circuit) and therefore we can safely change the garbled output gates of $\mathsf{C_{passive}}$ from $\mathsf{InputDepSimGate}$ to $\mathsf{SimGate}$.

3. We change the passive garbled circuit $\mathsf{C_{passive}}$ and the garbled input for it from being simulated to being a garbling of C_1, x_1. This follows from the same argument as the previous step.

4. We now modify the s-gates one-by-one to output the value of the passive circuit instead of the active circuit. This is the most delicate part of the proof. It essentially follows via a sequence of steps where, for each output $i \in [m]$, we use the proof strategy of [HJO+16] to change the i'th output gate of both $\mathsf{C_{active}}, \mathsf{C_{passive}}$ to be in $\mathsf{InputDepSimGate}$ mode. This means that these garbled gates aren't really created until the on-line phase when the garbled input is given out. Furthermore, when they are created in the on-line phase, each of these garbled gates only contains one key for the output wire corresponding to the correct bit going over that wire during the computation (either both corresponding to 0 or both to 1 since $C_0(x_0) = C_1(x_1)$). This allows us to change the encrypted value in 2 out 4 of the ciphertexts in the garbled s-gate so as to switch it from outputting the value of the active circuit to the one of the passive circuit.

5. We now repeat steps 2 and 3 for $\mathsf{C_{active}}$ to switch it from a garbling C_0, x_0, to simulated, to a garbling of C_1, x_1. Finally, we are left with the original security game with the challenge bit $b = 1$.

The above steps – except for step 4 – rely on the adaptive security of the underlying garbling scheme in a blackbox manner. It remains an open problem whether it is possible to show a more general transformation

from garbled circuits with adaptive security (and maybe other natural properties) to garbled circuits with indistinguishability based adaptive security and online complexity independent of the output size.

2 Preliminaries

General Notation. For a positive integer n, we define the set $[n] := \{1, \ldots, n\}$. We use the notation $x \leftarrow X$ for the process of sampling a value x according to the distribution X. For a vector $\overline{m} = (m_1, m_2, \cdots, m_n)$, and a subset $P \subset [n]$, we use $(m_i)_{i \in P}$ to denote a vector containing only the values m_i in positions $i \in P$ and $\perp$ symbols in all other positions. We use $(m_i)_{i \notin P}$ as shorthand for $(m_i)_{i \in [n] \setminus P}$.

Circuit Notation. A boolean circuit C consists of gates $\mathsf{gate}_1, \ldots, \mathsf{gate}_q$ and wires $w_1, w_2, \ldots, w_p$. A gate is defined by the tuple $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ where $g : \{0,1\}^2 \to \{0,1\}$ is the function computed by the gate, w_a, w_b are the incoming wires, and w_c is the outgoing wire. Although each gate has a unique outgoing wire w_c, this wire can be used as an incoming wire to several different gates and therefore this models a circuit with fan-in 2 and unbounded fan-out. We let q denote the number of gates in the circuit, n denotes the number of input wires and m denote the number of output wires. The total number of wires is $p = n + q$ (since each wire can either be input wire or an outgoing wire of some gate). For convenience, we denote the n input wires by $\mathsf{in}_1, \ldots, \mathsf{in}_n$ and the m output wires by $\mathsf{out}_1, \ldots, \mathsf{out}_m$. For $x \in \{0,1\}^n$ we write $C(x)$ to denote the output of evaluating the circuit C on input x.

Definition 1. Two distributions X and Y are (T, ε)-indistinguishable, denote $\mathbf{D}_T[X, Y] = \varepsilon$ if for any probabilistic algorithm $\mathcal{A}$, running in time T,

$$|\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]| \leq \varepsilon.$$

For two games $\textsc{Game}$ and $\textsc{Game}'$ we say they are $(T(\lambda), \varepsilon(\lambda))$- indistinguishable, $\mathbf{D}_{T(\lambda)}[\textsc{Game}, \textsc{Game}'] = \varepsilon(\lambda)$, if for any adversary $\mathcal{A}$ running in time $T(\lambda)$,

$$\left|\Pr[\textsc{Game}_{\mathcal{A}} = 1] - \Pr\left[\textsc{Game}'_{\mathcal{A}} = 1\right]\right| \leq \varepsilon(\lambda).$$

Let games $\textsc{Game}(\lambda)$ and $\textsc{Game}'(\lambda)$ be parametrized by the security parameter λ. If for any polynomial function $T(\lambda)$, there exists a negligible function $\varepsilon(\lambda)$, such that for all λ, $\mathbf{D}_{T(\lambda)}[\textsc{Game}(\lambda), \textsc{Game}'(\lambda)] \leq \varepsilon(\lambda)$, we say the two games are computationally indistinguishable and denote this by $\textsc{Game}(\lambda) \overset{\text{comp}}{\approx} \textsc{Game}'(\lambda)$.

We say C is leveled, if each gate has an associated level and any gate at level l has incoming wires only from gates at level $l-1$ and outgoing wires only to gates at level $l+1$. We let the *depth* d denote the number of levels and the *width* w denote the maximum number of gates in any level.

A circuit C is fully specified by a list of gate tuples $\mathsf{gate}_i = (g, w_a, w_b, w_c)$. We use $\Phi(C)$ to refer to the topology of a circuit– which indicates how gates are connected, without specifying the function implement by each gate. In other words, $\Phi(C)$ is the list of *sanitized gate tuples* $\widehat{\mathsf{gate}}_i = (\perp, w_a, w_b, w_c)$ where the function g that the gate implements is removed from the tuple.

3 Definitions

The bulk of this section defining what garbled circuits are and presenting Yao's construction, is taken verbatim from [HJO+16]. We now give a formal definition of a garbling scheme. There are many variants of such definitions in the literature, and we refer the reader to [BHR12] for a comprehensive treatment.

Definition 2. A Garbling Scheme is a tuple of PPT algorithms $\mathsf{GC} = (\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$ such that:

- $(\widetilde{C}, k) \xleftarrow{\$} \mathsf{GCircuit}(1^\lambda, C)$: takes as input a security parameter λ, a circuit $C : \{0,1\}^n \to \{0,1\}^m$, and outputs the *garbled circuit* $\widetilde{C}$, and *key* k.
- $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$: takes as input, $x \in \{0,1\}^n$, and key k and outputs $\tilde{x}$.

- $y = \mathsf{Eval}(\widetilde{C}, \tilde{x})$: given a garbled circuit $\widetilde{C}$ and a garbled input $\tilde{x}$ output $y \in \{0,1\}^m$.

Correctness There is a negligible function ν such that for any $\lambda \in \mathbb{N}$, any circuit C and input x it holds that $\Pr[C(x) = \mathsf{Eval}(\widetilde{C}, \tilde{x})] = 1 - \nu(\lambda)$, where $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$, $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$.

Adaptive Security (Based on Simulation). There exists a PPT simulator $\mathsf{Sim} = (\mathsf{SimC}, \mathsf{SimIn})$ such that, for any PPT adversary $\mathcal{A}$, there exists a negligible function ε such that:

$$\Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, 1) = 1] \leq \varepsilon(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(\lambda, b)$ is defined as follows:

1. The adversary $\mathcal{A}$ specifies C and gets $\widetilde{C}$ where $\widetilde{C}$ is created as follows:
   - if $b = 0$: $(\widetilde{C}, k) \leftarrow \mathsf{GCircuit}(1^\lambda, C)$,
   - if $b = 1$: $(\widetilde{C}, \mathsf{state}) \leftarrow \mathsf{SimC}(1^\lambda, \Phi(C))$,
2. The adversary $\mathcal{A}$ specifies x and gets $\tilde{x}$ created as follows:
   - if $b = 0$, $\tilde{x} \leftarrow \mathsf{GInput}(k, x)$,
   - if $b = 1$, $\tilde{x} \leftarrow \mathsf{SimIn}(C(x), \mathsf{state})$.
3. Finally, the adversary outputs a bit b', which is the output of the experiment.

In other words, we say GC is **adaptively secure** if

$$\mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Sim}}(\lambda, 0), \mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Sim}}(\lambda, 1)\right] = \varepsilon(\lambda).$$

Adaptive Security (Based on Indistinguishability). For any PPT adversary $\mathcal{A}$, there exists a negligible function ε such that:

$$\Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Ind}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Ind}}(\lambda, 1) = 1] \leq \varepsilon(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda, b)$ is defined as follows:

1. $\mathcal{A}$ specifies two circuits C_0, C_1 of the same topology, and gets back $\widetilde{C}_b \leftarrow \mathsf{GCircuit}(1^\lambda, C_b)$.
2. $\mathcal{A}$ specifies x_0, x_1 such that $C_0(x_0) = C_1(x_1)$ and gets $\tilde{x}_b \leftarrow \mathsf{GInput}(k, x_b)$.
3. Finally, the adversary outputs a bit b', which is the output of the experiment.

In other words, we say GC is **adaptively indistinguishable** if

$$\mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Ind}}(\lambda, 0), \mathsf{Exp}^{\mathsf{adaptive}}_{\mathsf{GC},\mathsf{Ind}}(\lambda, 1)\right] = \varepsilon(\lambda).$$

On-line Complexity. The time it takes to garble an input x, (i.e., time complexity of $\mathsf{GInput}(\cdot,\cdot)$) is the *on-line complexity* of the scheme. Clearly the on-line complexity of the scheme gives a bound on the size of the garbled input $\tilde{x}$. Ideally, the on-line complexity should be much smaller than the circuit size $|C|$.

Projective Scheme. We say a garbling scheme is *projective* if each bit of the garbled input $\tilde{x}$ only depends on one bit of the actual input x. In other words, each bit of the input, is garbled independently of other bits of the input. Projective schemes are essential for two-party computation where the garbled input is transmitted using an oblivious transfer (OT) protocol. Our constructions will be projective.

Hiding Topology. A garbling scheme that satisfies the above security definition may reveal the topology of the circuit C. However, there is a way to transform any such garbling scheme into one that hides everything, including the topology of the circuit, without a significant asymptotic efficiency loss. More precisely, we rely on the fact that there is a function $\mathsf{HideTopo}(\cdot)$that takes a circuit C as input and outputs a functionally equivalent circuit C', such that for any two circuits C_1, C_2 of equal size, if $C'_1 = \mathsf{HideTopo}(C_1)$ and $C'_2 = \mathsf{HideTopo}(C_2)$, then $\Phi(C'_1) = \Phi(C'_2)$. An easy way to construct such function $\mathsf{HideTopo}$ is by setting C' to be a universal circuit, with a hard-coded description of the actual circuit C. Therefore, to get a topology-hiding garbling scheme, we can simply use a topology-revealing scheme but instead of garbling the circuit C directly, we garble the circuit $\mathsf{HideTopo}(C)$.

4 Construction of [HJO+16]

In our construction (presented in the following section), we will use the construction of [HJO+16], as a building block. Furthermore we will need the details of this construction in order to proceed with the proof of security of our construction. Therefore in this section we present the construction of [HJO+16] which consists of two simple steps: (1) garble the circuit using Yao's garbling scheme; (2) hide the garbled circuit (without the output tables) under an **outer** layer of encryption instantiated with a *somewhere-equivocal* encryption scheme. In the on-line phase, the garbled input consists of Yao's garbled input plus the output tables. Next we provide the formal description of the scheme of [HJO+16] which contains the details of Yao's garbling scheme.

Let C be a leveled boolean circuit with fan-in 2 and unbounded fan-out, with inputs size n, output size m, depth d and width w. Let q denote the number of gates in C. Recall that wires are uniquely identified with labels $w_1, w_2, \ldots, w_p$, and a circuit C is specified by a list of gate tuples $\mathsf{gate} = (g, w_a, w_b, w_c)$. The topology of the circuit $\Phi(C)$ consists of the sanitized gate tuples $\widehat{\mathsf{gate}}_i = (\perp, w_a, w_b, w_c)$. For simplicity, we implicitly assume that $\Phi(C)$ is public and known to the circuit evaluator without explicitly including it as part of the garbled circuit $\widetilde{C}$. To simplify the description of our construction, we first describe the procedure for garbling a single gate, that we denote by $\mathsf{GarbleGate}$.

Let $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure symmetric-key encryption scheme satisfying the special correctness property defined in Appendix A. $\mathsf{GarbleGate}$ is defined as follows.

- $\widetilde{g} \leftarrow \mathsf{GarbleGate}(g, \{k_a^\sigma, k_b^\sigma, k_c^\sigma\}_{\sigma\in\{0,1\}})$: This function computes 4 ciphertexts $c_{\sigma_0,\sigma_1} : \sigma_0, \sigma_1 \in \{0,1\}$ as defined below and outputs them in a random order as $\widetilde{g} = [c_1, c_2, c_3, c_4]$.

$$c_{0,0} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{g(0,0)})) \quad c_{0,1} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{g(0,1)}))$$
$$c_{1,0} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{g(1,0)})) \quad c_{1,1} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{g(1,1)}))$$

Let $\Pi = (\mathsf{seKeyGen}, \mathsf{seEnc}, \mathsf{seDec}, \mathsf{SimEnc}, \mathsf{SimKey})$ be a somewhere-equivocal symmetric-encryption scheme as defined in Appendix B. Recall that in this primitive the plaintext is a vector of n blocks, each of which has s bits. In this construction the following parameters are used: the vector size $n = q$ is the number of gates and the block size $s = |\widetilde{g}|$ is the size of a single garbled gate. The equivocation parameter t is defined by the strategy used in the security proof and will be specified later. The garbling scheme is formally described in Fig. 1.

4.1 Adaptive Simulator

The adaptive security simulator for [HJO+16] is essentially the same as the selective security simulator for Yao's scheme (as in [LP09]), with the only difference that the output table is sent in the on-line phase, and is computed adaptively to map to the correct output. Note that the garbled circuit simulator does not rely on the simulation properties of the somewhere equivocal encryption scheme - these are only used in the proof of indistinguishability.

More specifically, the adaptive simulator $(\mathsf{SimC}, \mathsf{SimIn})$ works as follows. In the off-line phase, SimC computes the garbled gates using procedure $\mathsf{GarbleSimGate}$, that generates 4 ciphertexts that encrypt the same output key.

$\underline{\mathsf{GCircuit}(1^\lambda, C)}$

1. Garble Circuit (Yao's scheme)
   - (Wires) $k^\sigma_{w_i} \leftarrow \mathsf{Gen}(1^\lambda)$ for $i \in [p]$, $\sigma \in \{0,1\}$. (Input wires) $K = \left(k^0_{\mathsf{in}_i}, k^1_{\mathsf{in}_i}\right)_{i\in[n]}$.
   - (Gates) For each $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ in C: $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}\left(g, \left\{k^\sigma_{w_a}, k^\sigma_{w_b}, k^\sigma_{w_c}\right\}_{\sigma\in\{0,1\}}\right)$.
   - (Output tables) For each output $j \in [m]$: $\widetilde{d}_j := \left[\left(k^0_{\mathsf{out}_j} \to 0\right), \left(k^1_{\mathsf{out}_j} \to 1\right)\right]$.
2. Outer Encryption
   - $\mathsf{key} \xleftarrow{\$} \mathsf{seKeyGen}(1^\lambda)$.
   - $\widetilde{C} \leftarrow \mathsf{seEnc}\left(\mathsf{key}, (\widetilde{g}_1, \ldots, \widetilde{g}_q)\right)$.

Output $\widetilde{C}$, $k = \left(K, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]}\right)$.

$\underline{\mathsf{GInput}(x, k)}$

- (Select input keys) $K^x = \left(k^{x_1}_{\mathsf{in}_1}, \ldots, k^{x_n}_{\mathsf{in}_n}\right)$.
- **Output** $\tilde{x} = \left(K^x, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]}\right)$.

$\underline{\mathsf{Eval}(\widetilde{C}, \tilde{x})}$

1. Parse $\tilde{x} = \left(K, \mathsf{key}, (\widetilde{d}_j)_{j\in[m]}\right)$.
2. Decrypt Outer Encryption $(\widetilde{g}_i)_{i\in q} \leftarrow \mathsf{seDec}\left(\mathsf{key}, \widetilde{C}\right)$.
3. Evaluate Circuit.
   - Parse $K = (k_{\mathsf{in}_1}, \ldots, k_{\mathsf{in}_n})$.
   - For each level $j = 1, \ldots, d$ and for each $\widehat{\mathsf{gate}}_i = (\bot, w_a, w_b, w_c)$ at level j:
   - Let $\widetilde{g}_i = [c_1, c_2, c_3, c_4]$;
   - For $\delta \in [4]$ let $k'_{w_c} \leftarrow \mathsf{Dec}_{k_{w_a}}\left(\mathsf{Dec}_{k_{w_b}}(c_\delta)\right)$ If $k'_{w_c} \neq \bot$ then set $k_{w_c} := k'_{w_c}$.
4. Decrypt output. For $j \in [m]$:
   - Parse $\widetilde{d}_j = [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$.
   - Set $y_j = b$ iff $k_{\mathsf{out}_j} = k^b_{\mathsf{out}_j}$.

Output $y_1, \ldots, y_m$.

Figure 1: Adaptively-secure Garbling Scheme.

More precisely,

- $\mathsf{GarbleSimGate}(\{k^\sigma_{w_a}, k^\sigma_{w_b}\}_{\sigma\in\{0,1\}}, k'_{w_c})$ takes both keys for input wires w_a, w_b and a single key for the output wire w_c, that we denote by k'_{w_c}. It then output $\widetilde{g}_c = [c_1, c_2, c_3, c_4]$ where the ciphertexts, arranged in random order, are computed as follows.

$$c_{0,0} \leftarrow \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^0_b}(k'_c)) \quad c_{1,0} \leftarrow \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^0_b}(k'_c))$$
$$c_{0,1} \leftarrow \mathsf{Enc}_{k^0_a}(\mathsf{Enc}_{k^1_b}(k'_c)) \quad c_{1,1} \leftarrow \mathsf{Enc}_{k^1_a}(\mathsf{Enc}_{k^0_b}(k'_c))$$

The simulator invokes $\mathsf{GarbleSimGate}$ on input $k'_c = k^0_c$. It then encrypts the garbled gates so obtained by using the honest procedure for the somewhere equivocal encryption.

In the on-line phase, SimIn, on input $y = C(x)$ adaptively computes the output tables so that the evaluator obtains the correct output. This is easily achieved by associating each bit of the output, y_j, to the only key encrypted in the output gate g_{out_j}, which is $k^0_{\mathsf{out}_j}$. For the input keys, SimIn just sends keys $k^0_{\mathsf{in}_i}$ for each $i \in [n]$. The detailed definition of $(\mathsf{SimC}, \mathsf{SimIn})$ is provided in Fig. 3.

5 Our Construction

Let $\mathsf{cGC} = (\mathsf{cGCircuit}, \mathsf{cGInput}, \mathsf{cEval})$ be the adaptive garbling scheme of [HJO+16], with simulator $\mathsf{cSim} = (\mathsf{cSimC}, \mathsf{cSimIn})$. In this section we construct a new garbling scheme, using cGC as a building block. See Figure 5 for a formal description of our construction. The new garbling scheme creates two copies of the garbled circuit (called $\mathsf{C_L}, \mathsf{C_R}$). It chooses one at random to be the "active" one ($\mathsf{active} = R$ or $\mathsf{active} = L$). Then for each output bit $i \in [m]$, it creates a selection gate that takes the output wire i from both garbled circuits, and selects the value on the wire coming from the active circuit. We call these selection gates,

Simulator

$\underline{\mathsf{SimC}(1^\lambda, \Phi(C))}$

- (Wires) $k^\sigma_{w_i} \leftarrow \mathsf{Gen}(1^\lambda)$ for $i \in [p]$, $\sigma \in \{0,1\}$.
- (Garbled gates) For each gate $\widetilde{\mathsf{gate}}_i = (\bot, w_a, w_b, w_c))$ in $\Phi(C)$: $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}\left(k^\sigma_{w_a}, k^\sigma_{w_b}\}_{\sigma\in\{0,1\}}, k^0_{w_c}\right)$.
- (Outer Encryption): $\mathsf{key} \xleftarrow{\$} \mathsf{seKeyGen}(1^\lambda)$, $\widetilde{C} \leftarrow \mathsf{seEnc}\left(\mathsf{key}, \widetilde{g}_1, \ldots, \widetilde{g}_q\right)$.
- **Output** $\widetilde{C}$, $\mathsf{state} = \left(\{k^\sigma_{w_i}\}, \mathsf{key}\right)$.

$\underline{\mathsf{SimIn}(y, \mathsf{state})}$

- Generate output table: $\widetilde{sd}_j \leftarrow \left[\left(k^{y_j}_{\mathsf{out}_j} \to 0\right), \left(k^{1-y_j}_{\mathsf{out}_j} \to 1\right)\right]_{j\in[m]}$. // ensures $k^0_{\mathsf{out}_j} \to y_j$
- **Output** $\widetilde{x} = \left(\left(k^0_{\mathsf{in}_i}\right)_{i\in[n]}, \mathsf{key}, \left(\widetilde{sd}_j\right)_{j\in[m]}\right)$.

Figure 2: Simulator for Adaptive Security.

sgate_R	sgate_L
$\mathsf{Enc}_{\ell^0}(\mathsf{Enc}_{r^1}(1))$	$\mathsf{Enc}_{\ell^0}(\mathsf{Enc}_{r^1}(0))$
$\mathsf{Enc}_{\ell^0}(\mathsf{Enc}_{r^0}(0))$	$\mathsf{Enc}_{\ell^0}(\mathsf{Enc}_{r^0}(0))$
$\mathsf{Enc}_{\ell_1}(\mathsf{Enc}_{r^1}(1))$	$\mathsf{Enc}_{\ell^1}(\mathsf{Enc}_{r^1}(1))$
$\mathsf{Enc}_{\ell^1}(\mathsf{Enc}_{r^0}(0))$	$\mathsf{Enc}_{\ell^1}(\mathsf{Enc}_{r^0}(1))$

Figure 3: s-gates. $\mathsf{sgate}_L(\mathsf{sgate}_R)$ outputs the value associated with the wire coming form $\mathsf{C_L}, (\mathsf{C_R})$.

s-gates, to distinguish them from the output gates of the two original garbled circuits. Let ℓ^b and r^b be the output wires of $\mathsf{C_L}$ and $\mathsf{C_R}$, then s-gate (for each output bit) is defined as in Figure 4.

Note that $\mathsf{C_{active}}$ and $\mathsf{C_{passive}}$ are encrypted Yao garbled circuits. But the output wires and the output map are not encrypted and are part of the key k which is an output of $\mathsf{cGCircuit}(\cdot,\cdot)$.

6 Hybrid Games

Overview. We need to prove that $\textsc{Game}_0 = \mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{NGC},\mathsf{Ind}}(\lambda, 0)$ and $\textsc{Game}_1 = \mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{NGC},\mathsf{Ind}}(\lambda, 1)$ are indistinguishable. Namely, we need to show a strategy to move from $\textsc{Game}_0$, where ($\mathsf{C_{passive}}$, $\mathsf{C_{active}}$) are both garbling of C_0 and (x_{active}, x_{passive}) are garblings of x_0; to $\textsc{Game}_1$ where ($\mathsf{C_{passive}}$, $\mathsf{C_{active}}$) are garbling of C_1 and (x_{active}, x_{passive}) are garblings of x_1.

At high-level, the proof strategy is the following: starting from $\textsc{Game}_0$, (1) first we change $\mathsf{C_{passive}}, x_{\mathsf{passive}}$ to be the garbling of C_1, x_1, (2) then we change the selection gates so that they select outputs from $\mathsf{C_{passive}}$, (3) finally we change $\mathsf{C_{active}}, x_{\mathsf{active}}$ to be the garbling of C_1, x_1.

For step (1) and (3), we switch from garbling C_0, x_0 to garbling C_1, x_1 by using simulated circuits, namely first we change $\mathsf{C_{passive}}$ into a simulated circuit, and then we switch it into a real garbling of C_1. Indistinguishability of this steps follows directly from the adaptive simulation-based security of the underlying garbling scheme in a black-box manner (we discuss this next in Sec. 6.1). Changing the selection gates (Step 2) instead requires a surgical proof, where we selective simulate one *output gate* of $\mathsf{C_{passive}}, \mathsf{C_{active}}$ at the time, and this enable us to change (switch) the content of the selection gates, from selecting the output of $\mathsf{C_{passive}}$ instead of $\mathsf{C_{active}}$ (or viceversa). Following the language of [HJO+16], this means that we need to place black pebbles on the output gates of circuits $\mathsf{C_{passive}}, \mathsf{C_{active}}$. We discuss this in details in Lemma 3.

N Garbling Scheme

$\mathsf{NGCircuit}(1^\lambda, C)$.

1. $\mathsf{active} \leftarrow \{L, R\}$. If $\mathsf{active} = L$ then $\mathsf{passive} = R$ else $\mathsf{passive} = L$.
2. $(\mathsf{C_L}, k_L) \leftarrow \mathsf{cGCircuit}(1^\lambda, C)$ and $(\mathsf{C_R}, k_R) \leftarrow \mathsf{cGCircuit}(1^\lambda, C)$
3. Parse k_α into $\left(K_\alpha, \mathsf{key}_\alpha, (\widetilde{cd}_{\alpha,i})_{i\in[m]}\right)$ for $\alpha \in \{L, R\}$
4. For $i \in [m]$ let sgate_i computed as $\mathsf{sgate}_{\mathsf{active}}$ (Figure 4) with the ith output wire of $\mathsf{C_R}$ and $\mathsf{C_L}$ as input. Let $\widetilde{\mathsf{sgate}} = (\mathsf{sgate}_1, \ldots, \mathsf{sgate}_m)$
5. $\widetilde{C} := \left(\mathsf{C_L}, \mathsf{C_R}, \widetilde{\mathsf{sgate}}\right)$.
6. $k_L := (K_L, \mathsf{key}_L)$, $k_R := (K_R, \mathsf{key}_R)$, $k := (k_L, k_R)$.
7. Output $\widetilde{C}, k$.

$\mathsf{NGInput}(x, k)$

1. (select keys) $K_L^x = \mathsf{SelGInput}(x, K_L)$ and $K_R^x = \mathsf{SelGInput}(x, K_R)$.
2. $\tilde{x}_L = (K_L^x, \mathsf{key}_L), \tilde{x}_R = (K_R^x, \mathsf{key}_R)$
3. Output $\tilde{x} = (\tilde{x}_L, \tilde{x}_R)$

$\mathsf{NEval}(\widetilde{C}, \tilde{x})$

1. $\{w_{\alpha,i}\}_{i\in[m]} := \mathsf{cEval}(\mathsf{C}_\alpha, \tilde{x}_\alpha)$, for $\alpha \in \{L, R\}$
2. Parse $\mathsf{sgate}_1, \ldots, \mathsf{sgate}_m \leftarrow \widetilde{\mathsf{sgate}}$.
3. Use keys $\{w_{\alpha,i}\}_{i\in[m]}$ to evaluate gates $\mathsf{sgate}_1, \ldots, \mathsf{sgate}_m$ and obtain y.
4. Output y.

Figure 4: New garbling scheme

6.1 Hybrid Games Template

The hybrid games are parameterized by the distributions of $\mathsf{C}_{\mathsf{active}}$, $\mathsf{C}_{\mathsf{passive}}$, their respective inputs $x_{\mathsf{active}}, x_{\mathsf{passive}}$ and a flag $\alpha \in \{\mathsf{active}, \mathsf{passive}\}$ denoting the fact that s-gates are selecting the ouput of C_α

For example the original $\textsc{Game}_b$ is described as:

- $\textsc{Game}_0 = \left(\left(\mathsf{cGCircuit}(1^\lambda, C_0), x_0\right), \left(\mathsf{cGCircuit}(1^\lambda, C_0), x_0\right)\right), \mathsf{active}\right)$
- $\textsc{Game}_1 = \left(\left(\mathsf{cGCircuit}(1^\lambda, C_1), x_1\right), \left(\mathsf{cGCircuit}(1^\lambda, C_1), x_1\right)\right), \mathsf{active}\right)$

Note that when the active and passive garbled circuit distributions are the same, it does not make a difference whether $\alpha = \mathsf{active}$ or $\alpha = \mathsf{passive}$. However in our hybrid argument we will sometimes set $\alpha = \mathsf{passive}$ when these distributions are different. We use $\mathsf{cSimC}(1^\lambda, \Phi(C))$ to denote a simulated circuit. Since the simulated garbling of any circuit only depends on its topology and not the function it computes, the output of the simulation has the same distribution for C_0 and C_1, thus for simplicity we write $\mathsf{cSimC}(1^\lambda, \Phi(C))$.

Using this template we define 4 new hybrid games: HybA through HybD. See Figure 6. The changes in these hybrids follow a two-step simulate and switch approach. In HybA the passive circuit is simulated. Note that the garbled input to a simulated circuit is created independent of the input, therefore its distribution does not change whether it's x_0 that is garbled or x_1. In HybB the passive circuit is switched from simulation to real

garbling of C_1. Now with both active and passive circuits outputing the same value $y = C_0(x_0) = C_1(x_1)$, we go to the next hybrid. In HybC we change the content of the s-gates to output the passive circuit. Then we turn the active circuit into a garbling of C_1 with input x_1, by first simulating it (HybD) and then changing it to a garbling of C_1 with input x_1 (GAME_1). The transitions from GAME_0 to HybA then to HybB are identical to the ones going from GAME_1 to HybD and then to HybC. Thus we only prove it once for $\text{GAME}_0 \overset{\text{comp}}{\approx} \mathsf{HybA} \overset{\text{comp}}{\approx} \mathsf{HybB}$.

Hybrids	GAME_0	HybA	HybB
$\mathsf{C}_{\mathsf{active}}, x_{\mathsf{active}}$ $\mathsf{C}_{\mathsf{passive}}, x_{\mathsf{passive}}$ sgate outputs	$\mathsf{cGCircuit}(1^\lambda, C_0), x_0$ $\mathsf{cGCircuit}(1^\lambda, C_0), x_0$ active	$\mathsf{cGCircuit}(1^\lambda, C_0),\ x_0$ $\mathsf{cSimC}(1^\lambda, \Phi(C)),\ x_1$ active	$\mathsf{cGCircuit}(1^\lambda, C_0),\ x_0$ $\mathsf{cGCircuit}(1^\lambda, C_1),\ x_1$ active
Hybrids	HybC	HybD	GAME_1
$\mathsf{C}_{\mathsf{active}}, x_{\mathsf{active}}$ $\mathsf{C}_{\mathsf{passive}}, x_{\mathsf{passive}}$ sgate outputs	$\mathsf{cGCircuit}(1^\lambda, C_0), x_0$ $\mathsf{cGCircuit}(1^\lambda, C_1), x_1$ passive	$\mathsf{cSimC}(1^\lambda, \Phi(C)), x_1$ $\mathsf{cGCircuit}(1^\lambda, C_1), x_1$ passive	$\mathsf{cGCircuit}(1^\lambda, C_1), x_1$ $\mathsf{cGCircuit}(1^\lambda, C_1), x_1$ passive

Figure 5: Hybrids.

6.1.1 From Game$_0$ to HybA.

To prove this, we are going to need a special property that is enjoyed by the garbling scheme cGC. We define the special property below.

Definition 3 (Output-key Security). We say that an adaptively simulation-secure garbling scheme is *output-key* secure if it is adaptively secure even when the output keys (e.g., $\{w_{\alpha,i}\}_{i\in[m]}$) –without the output mapping– are sent together with the garbled circuit $\widetilde{C}$.

Proposition 1. Under the same assumptions as [HJO+16], the garbling scheme cGC is adaptively secure and output-key secure.

[Proof Sketch]. Intuitively this is true because throughout the proof of security for cGC we rely on the CPA security of the encryption scheme used to garble the gates, to prove the adversary does not learn the content of any gates, before getting the garbled input, and even after seeing the garbled input he can only decipher one ciphertext from each garbled gate. During these reductions, we can even let the adversary choose the keys encrypted in a garbled output gate (as in the game for the CPA security, the adversary can choose any message to be encrypted). Furthermore the output keys are not used as an encryption key somewhere else in the same garbled circuit, therefore revealing the output key does not jeopardize the adaptive security of cGC.

Now that we have defined the property above, we can prove the following Lemma.

Lemma 1. If cGC is adaptively secure and output-key secure, then GAME_0 and HybA are computationally indistinguishable.

Proof. If a PPT adversary $\mathcal{A}$ distinguishes GAME_0 and HybA with advantage ε, we construct adversary $\mathcal{B}$ that breaks the adaptive security of cGC with the same advantage ε. $\mathcal{B}$ will receive C_0, C_1 from $\mathcal{A}$, and sends C_0 to its challenger, and gets back $\widetilde{C}^*$, which is $(\widetilde{C}^*, k) \leftarrow \mathsf{cGCircuit}(1^\lambda, C_0)$ if $b = 0$ and$(\widetilde{C}^*, \mathsf{state}) \leftarrow \mathsf{cSimC}(1^\lambda, \Phi(C))$ if $b = 1$. $\mathcal{B}$ then sets $(\mathsf{C}_{\mathsf{active}}, k_0) \leftarrow \mathsf{cGCircuit}(1^\lambda, C_0)$ and $\mathsf{C}_{\mathsf{passive}} = \widetilde{C}^*$. Next, $\mathcal{B}$ creates the s-gates so that they would reveal the output of $\mathsf{C}_{\mathsf{active}}$. Note that $\mathcal{B}$ does not need the output map of $\widetilde{C}^*$ to create s-gates, it only needs the keys encrypted in the output level gates of $\widetilde{C}^*$. Which we assume are given as part of the garbled circuit, without jeopardizing the security of cGC (due to output-key security).

Finally $\mathcal{B}$ sends $\widetilde{C} = \left(\mathsf{C_L}, \mathsf{C_R}, \widetilde{\mathsf{sgate}}\right)$ to $\mathcal{A}$ and gets back x_0, x_1. $\mathcal{B}$ sends x_0 to the challenger and gets back $\tilde{x}^*$ which is $\tilde{x}^* \leftarrow \mathsf{cGInput}(x_0, k)$ if $b = 0$ and $\tilde{x}^* \leftarrow \mathsf{SimIn}(C_0(x_0), \mathsf{state})$ if $b = 1$. The reduction will set $\tilde{x}_{\mathsf{active}} \leftarrow \mathsf{cGInput}(x_0, k_{\mathsf{active}})$, $\tilde{x}_{\mathsf{passive}} = \tilde{x}^*$ and sends $(\tilde{x}_L, \tilde{x}_R)$ to $\mathcal{A}$ and outputs $\mathcal{A}$'s final output, b'. Note, since SimIn does not even take in the input x_1 or x_0, it only gets the output of the computation in order to create the appropriate output map. And in this application, the output wires are treated the same way, regardless of whether they are mapped to 0 or 1, it doesn't matter which input is garbled by the simulator. □

Reduction $\mathcal{B}$

1. Receive C_0, C_1 from $\mathcal{A}$.
2. $\mathsf{active} \leftarrow \{L, R\}$. If $\mathsf{active} = L$ then $\mathsf{passive} = R$ else $\mathsf{passive} = L$.
3. Send C_0 to the challenger and get back $\widetilde{C}^*$.
4. Follow the steps for creating $\mathsf{NGCircuit}(1^\lambda, C_0)$ with one exception; use $\widetilde{C}^*$ as $\mathsf{C_{passive}}$.
5. Send $\widetilde{C} := \left(\mathsf{C_L}, \mathsf{C_R}, \widetilde{\mathsf{sgate}}\right)$ to $\mathcal{A}$ and receive x_0, x_1 .
6. Send x_0 to the challenger and get back $\tilde{x}^*$.
7. (select keys) $K^{x_0} = \mathsf{SelGInput}(x_0, K_{\mathsf{active}})$.
8. $\tilde{x}_{\mathsf{active}} = \left(K^{x_0}, \mathsf{key}_{\mathsf{active}}\right), \tilde{x}_{\mathsf{passive}} = \tilde{x}^*$
9. Send $\tilde{x} = (\tilde{x}_L, \tilde{x}_R)$ to $\mathcal{A}$ and receive b' from $\mathcal{A}$
10. Output b'

Figure 6: Reduction of Lemma 1

Lemma 2. If cGC is adaptively secure and output-key secure, then HybA and HybB are computationally indistinguishable.

Proof. It follows from a similar reduction to the one used in the proof of Lemma 1, with the difference that C_1, x_1 are sent to the challenger instead of C_0, x_0. □

Lemma 1 and Lemma 2 prove that:

$$\textsc{Game}_0 \overset{\text{comp}}{\approx} \mathsf{HybA} \overset{\text{comp}}{\approx} \mathsf{HybB} \text{ and } \mathsf{HybC} \overset{\text{comp}}{\approx} \mathsf{HybD} \overset{\text{comp}}{\approx} \textsc{Game}_1.$$

6.1.2 From HybB to HybC

Recall the distribution of hybrid HybB and HybC

- $\mathsf{HybB} = \left(\left(\mathsf{cGCircuit}(1^\lambda, C_0), x_0\right), \left(\mathsf{cGCircuit}(1^\lambda, C_1), x_1\right)\right), \mathsf{active}\right)$
- $\mathsf{HybC} = \left(\left(\mathsf{cGCircuit}(1^\lambda, C_0), x_0\right), \left(\mathsf{cGCircuit}(1^\lambda, C_1), x_1\right)\right), \mathsf{passive}\right)$

The difference between these two hybrids is only in the s-gates: instead of selecting the output from $\mathsf{C_{active}}$ (in HybB), now s-gates will select the output from $\mathsf{C_{passive}}$ (in HybC). Recall the description of s-gate in Fig. 4. Changing the s-gates from active to passive entails changing 2 of the encryptions. In order to argue that these changes are indistinguishable, we must rely on the CPA security of the encryption. However the keys used to create these ciphertexts are not *independent*, since they are used in the garbling of the output gates of $\mathsf{C_L}$ and $\mathsf{C_R}$. Therefore, if we want to change even one encryption, we need to *remove* those keys from

the correspondent gates in C_L and C_R. In other words, those two gates need to be *simulated*. Now, in order to change one gate at the time from real to simulated, we need to leverage the details of the proof provided in [HJO+16].

Proof Strategy in [HJO+16]. We now give an overview of the proof strategy of [HJO+16]; we rely on specific components of the strategy in our proof. For more details see Appendix C. In [HJO+16] hybrid games are parametrized by a *circuit configuration*, that is, a vector indicating the way the gates are garbled. There are three modes for how each gate can be garbled: RealGate, InputDepSimGate, SimGate. There are also *rules* that allow one to indistinguishably move from one configuration to another. These configurations/rules are summarized via a pebbling game where we associate RealGate mode to a gate *not having a pebble on it*, InputDepSimGate mode is associated with a gate having a *black pebble*, and SimGate mode is associated with a gate having a *grey pebble*. The indistinguishability *rules* are then translated to rules for the pebbling game:

Pebbling Rule A. We can place or remove a black pebble on a gate as long as both predecessors of that gate have black pebbles on them (or the gate is an input gate).

Pebbling Rule B. We can replace a black pebble with a grey pebble on a gate as long as all successors of that gate have black or grey pebbles on them (or the gate is an output gate).

We can follow the same rules for the two garbled circuits $\mathsf{C}_{\mathsf{active}}$, $\mathsf{C}_{\mathsf{passive}}$ with one major difference: we cannot replace a black pebble with a grey pebble on the output gates (this part relied on the fact that the output map, which specified the correspondence between wire keys at the output level and the bits they correspond to, was only sent in the on-line phase; in our case this correspondence is needed to create the s-gates in the off-line phase, at least for the active circuit).

We rely on one more property (*): if a gate has an output wire w which is associated with keys k_w^0, k_w^1 and we garble the gate in InputDepSimGate mode then we only use one key (k_w^b where b is the bit that the wire takes on during the computation $C(x)$) when creating this garbled gate in the on-line phase.

Let us define $\mathcal{C}[\gamma, t]$ to be the class of circuits C such that we can place a black pebble on any single output gate of C in γ pebbling steps and using at most t black pebbles at each step. For the following lemma, theorem and corollaries, assume:

1. The adversary selects $C_0, C_1 \in \mathcal{C}[\gamma, t]$.
2. $\Pi = (\mathsf{seKeyGen}, \mathsf{seEnc}, \mathsf{seDec}, \mathsf{SimEnc}, \mathsf{SimKey})$ is a somewhere equivocal encryption scheme with equivocation parameter t.
3. $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an encryption scheme secure under *chosen double encryption*.

Lemma 3. HybB and HybC are computationally indistinguishable.

Proof. Let m be the output size of the circuits C_0, C_1 selected by the adversary. For $i = 1, \ldots, m$, we rely on the following sequence of sub-hybrids:

1. Via a sequence of sub-sub-hybrids, change the configurations of both $\mathsf{C}_{\mathsf{active}}$ and $\mathsf{C}_{\mathsf{passive}}$ so that the i'th output gate is in InputDepSimGate mode (has a black pebble on it). This follows using the same argument as in [HJO+16].
2. Change the i'th s-gate from $\mathsf{sgate}_{\mathsf{active}}$ to $\mathsf{sgate}_{\mathsf{passive}}$ (see Figure 4). This change relies on property (*) and the CPA-security of the encryption scheme Γ used to garble the gates. In particular, this change requires changing the contents of the ciphertexts $\mathsf{Enc}_{\ell^0}(\mathsf{Enc}_{r^1}(?))$ and $\mathsf{Enc}_{\ell_1}(\mathsf{Enc}_{r^0}(?))$ in s-gate. However, since $C_0(x_0) = C_1(x_1)$ by property (*) the only keys that are used as plaintexts in other garbled gates in this hybrid are either (ℓ^0, r^1) or (ℓ^1, r^0). In either case, we can rely on encryption security to change the contents of the above two ciphertexts.
3. Via a sequence of sub-sub-hybrids, change the configurations of both $\mathsf{C}_{\mathsf{active}}$ and $\mathsf{C}_{\mathsf{passive}}$ back so that all gates are in RealGate mode (no pebbles). This is the same as step 1 in reverse.

□

From lemmas 1,2, 3, it follows that GAME_0 and GAME_1 are computationally indistinguishable which proves our main result, summarized in the following theorem.

Theorem 1. *Assuming the existence of one-way functions,* NGC *is adaptively indistinguishable with online complexity* $(n+t)\mathsf{poly}(\lambda)$ *for all circuits in* $\mathcal{C}[\mathsf{poly}(\lambda), t]$.

Using the pebbling strategies from [HJO+16] summarized in Appendix D we get the following bounds.

Lemma 4. Any circuit C of depth d, width w, with input size n and output size m, is in the class $\mathcal{C}[\gamma, t]$ with either of the following two settings of γ, t:

- $\gamma = 2^{(2d+1)}m$ steps using $t = 2d$ black pebbles.
- $\gamma = 4|C|$ steps using $t = 2w$ black pebbles.

Plugging the above lemma into Theorem 1 we get the following corollary.

Corollary 1. *Assuming the existence of one-way functions,* NGC *is adaptively indistinguishable with online complexity* $n \cdot \mathsf{poly}(\lambda)$ *for all circuits with either linear width* $w = O(n)$ *or logarithmic depth* $d = O(\log n)$.

Note that any computation which can be performed in linear space can be represented by a circuit with linear width. Therefore the above covers all linear space computations.

7 Application: Private-key Adaptively Secure Functional Encryption

Overview. Our new garbling scheme can be used to implement a private-key functional encryption ([SW05, BSW11]) based on one-way functions, with indistinguishability based security where the adversary can obtain an unbounded number of function secret keys and then adaptively a single challenge ciphertext (the formal definition is provided in Sec. 7.1).

In our scheme (described in Fig. 8), the functional keys are garbled circuits computed according to (a slightly modified version of) NGCircuit, and the ciphertext for a message m corresponds to the garbling of the input m. Since a single garbled input should be used to evaluate multiple garbled circuits, we slightly tweak the construction of our garbling scheme so to allow an initial state that is used upon each invocation of the garbling function. We explain this modification in greater length in Sec. 7.2.

7.1 Definition

A private-key functional encryption scheme Π, over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_\lambda$ and a circuit space $\mathcal{C} = \{C_\lambda\}_\lambda$ is a tuple of PPT algorithms (Π.FE.Setup, Π.FE.KeyGen, Π.FE.Enc, FE.Dec) defined as follows:

- $\Pi.\mathsf{FE.Setup}(1^\lambda)$: The setup algorithm takes as input the unary representation of the security parameter, and outputs a secret key MSK.
- $\Pi.\mathsf{FE.KeyGen}(\mathsf{MSK}, C)$: The key-generation algorithm takes as input a secret key MSK and a circuit $C \in \mathcal{C}_\lambda$ and outputs a functional key sk_C.
- $\Pi.\mathsf{FE.Enc}(\mathsf{MSK}, m)$: The encryption algorithm takes as input a secret key MSK and a message $m \in \mathcal{M}_\lambda$ and outputs a ciphertext CT.
- $\Pi.\mathsf{FE.Dec}(sk_C, \mathsf{CT})$ The decryption algorithm takes as input a functional key sk_C and a ciphertext CT, and outputs $m \in \mathcal{M}_\lambda \cup \{\perp\}$.

The correctness property requires that there exists a negligible function $negl(\cdot)$ such that for all sufficiently large $\lambda \in N$, for every message $m \in \mathcal{M}_\lambda$, and for every circuit $C \in \mathcal{C}_\lambda$ it holds that:

$$Pr[\mathsf{FE.Dec}(\Pi.\mathsf{FE.KeyGen}(\mathsf{MSK}, C), \mathsf{FE.Enc}(\mathsf{MSK}, m)) = C(m)] \geq 1 - negl(\lambda)$$

where $\mathsf{MSK} = \mathsf{FE.Setup}(1^\lambda)$ and the probability is taken over the random choices of all algorithms.

Many Functions Single Message Adaptive Security. For any PPT adversary $\mathcal{A}$, there exists a negligible function ε such that:

$$\Pr[\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda,0)=1]-\Pr[\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda,1)=1]\leq\varepsilon(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\,\mathsf{Ind}}(\lambda,b)$ is defined as follows:

1. **Query**. The adversary $\mathcal{A}$ specifies circuits $C^1, C^2, \ldots$. It then obtain functional keys $sk_1, sk_2, \ldots$ which are created as follow:
   - Run $\mathsf{MSK} = \Pi.\mathsf{FE.Setup}(1^\lambda)$.
   - Let q be the number of queries. $\forall i \in [q]$, $sk_i = \Pi.\mathsf{FE.KeyGen}(\mathsf{MSK}, C^i)$.
2. **Challenge**. The adversary $\mathcal{A}$ specifies messages m_0, m_1, such that for all $i \in [q]$, $C^i(m_0) = C^i(m_1)$ and obtains CT, which is created as follows:
   - $\mathsf{CT} = \Pi.\mathsf{FE.Enc}(\mathsf{MSK}, m_b)$
3. **Output**. Finally, the adversary outputs a bit b', which is the output of the experiment.

7.2 Construction

Our private-key functional encryption scheme is depicted in Figure 8. The $\mathsf{FE.Setup}$ algorithm generates the keys that need to be shared by all garbled circuits. Such keys are: (1) the keys for the input wires (i.e., K_L, K_R) (2) the keys for the outer somewhere-equivocal encryption seEnc (i.e., $\mathsf{key}_L, \mathsf{key}_R$). The $\mathsf{FE.Setup}$ also sets the flag active.

The $\mathsf{FE.KeyGen}$ algorithm generates a garbled circuit according to procedure $\mathsf{NGCircuit}^\star$ which is a slight modification of $\mathsf{NGCircuit}$ (shown in Figure 4) that enables to use a single garbled input to evaluate many garbled circuits generated at different times. The modifications are: (1) instead of running procedure $\mathsf{GCircuit}(1^\lambda, C)$ (described in Figure 1) – which would select fresh keys for the input wires and for the outer encryption – it runs a slightly modified procedure $\mathsf{GCircuit}^\star(1^\lambda, C, \mathit{Input\ keys})$ which takes such keys as an external input; (2) the encryption algorithm seEnc used in $\mathsf{GCircuit}$, is also slightly modified so that it allows blocks to be encrypted in a streaming fashion (that is, instead of having a one-time encryption of n blocks, we allow for many encryptions, where the total number of encrypted blocks is overall $\leq N$ where N is an upperbound (e.g., 2^λ)). In Appendix B we discuss why this modification (that we call $\mathsf{seEnc}^\star$) follows naturally from the implementation of seEnc provided in [HJO+16].

The $\mathsf{FE.Enc}$ algorithm takes in input a message m and simply runs the procedure $\mathsf{GInput}(m, \mathit{Input\ keys})$ to select the keys for m. The ciphertext then consists of the keys for the garbled inputs, and the keys for the outer encryption $\mathsf{key}_R, \mathsf{key}_L$. Note that the size of the ciphertex depends on the length of the input and the length of the keys $\mathsf{key}_R, \mathsf{key}_L$ for somewhere-equivocal encryption. Finally the decryption algorithm simply consists of the evaluation of the garbled circuits.

7.3 Security Proof

In this section we show that protocol in Figure 8 is a private-key functional encryption scheme that is adaptively secure for many function queries and a single message query (according to Definition 7.1).

Let $\textsc{Game}_0$, be the experiment $\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda,0)$ where the adversary receives encryption of m_0, and let $\textsc{Game}_1$ be the experiment $\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda,1)$. The proof of security consists of a sequence of hybrid games from $\textsc{Game}_0$ to $\textsc{Game}_1$, and each hybrid is computational indistinguishable. We now argue that this sequence of hybrids follows exactly the hybrids provided in the proof of Theorem 1.

Recall that in the security experiment $\mathsf{Exp}^{\mathsf{Private-FE}}_{\mathcal{A},\Pi,\mathsf{Ind}}(\lambda,b)$, $\mathcal{A}$ sends all function queries $C^1, C^2, \ldots, C^q$ at the beginning in one-shot. Concretely, by instantiating the experiment with Π, when $\mathcal{A}$ sends functional queries $C^1, C^2, \ldots, C^q$, she obtains:

Functional Keys: $\quad([\mathsf{C_L}^1, \mathsf{C_R}^1, \mathsf{SG}^1], \ldots, [\mathsf{C_L}^q, \mathsf{C_R}^q, \mathsf{SG}^q])$

where SG^j is the selection circuit $\widetilde{\mathsf{sgate}}$ associated to $\mathsf{C_L}^j, \mathsf{C_R}^j$.

Private-Key Functional Encryption

$\Pi.\mathsf{FE.Setup}(\lambda)$.

1. Select active garbled circuit.
   $\mathsf{active} \leftarrow \{L, R\}$. If $\mathsf{active} = L$ then $\mathsf{passive} = R$ else $\mathsf{passive} = L$.
2. Select keys for input wires:
   (left circuits) $K_L = \left(k^{0,\mathsf{a}}_{\mathsf{in}_i}, k^{1,\mathsf{a}}_{\mathsf{in}_i}\right)_{i\in[n]}$. (rigth circuits) $K_R = \left(k^{0,\mathsf{p}}_{\mathsf{in}_i}, k^{1,\mathsf{p}}_{\mathsf{in}_i}\right)_{i\in[n]}$.
   with $k^{\sigma,\alpha}_{w_i} \leftarrow \mathsf{Gen}(\lambda)$ for $i \in [n]$, $\sigma \in \{0,1\}$, $\alpha \in \{L, R\}$.
3. Select keys for outer encryption:
   (left/right circuits) $\mathsf{key}_L, \mathsf{key}_R$; where $\mathsf{key}_\alpha \xleftarrow{\$} \mathsf{seKeyGen}(\lambda)$, $\alpha \in \{L, R\}$.
4. **Output** $\mathsf{MSK} := \{K_L, \mathsf{key}_L, K_R, \mathsf{key}_R, \mathsf{active}\}$

$\Pi.\mathsf{FE.KeyGen}(\mathsf{MSK}, C)$.

1. $\widetilde{C} := \mathsf{NGCircuit}^\star(C, \mathsf{MSK})$
2. Ouput $sk_C = \widetilde{C}_b$.

$\Pi.\mathsf{FE.Enc}(\mathsf{MSK}, m)$.

1. $\tilde{x} = \mathsf{NGInput}(m, \mathsf{MSK})$.
2. Output $\mathsf{CT} = \tilde{x} = (K^x_L, \mathsf{key}_L, K^x_R, \mathsf{key}_R)$

$\mathsf{FE.Dec}(sk_C, \mathsf{CT})$.

1. Output $m = \mathsf{NEval}\,(sk_C, \mathsf{CT})$.

Figure 7: Private-Key FE

In the challenge phase, $\mathcal{A}$ receives the garbling of message m_b. Specifically:

Ciphertext: $\tilde{x} = (K_L, \mathsf{key}_L, K_R, \mathsf{key}_R)$

Now, note that, because the functional keys (i.e., the garbled circuits) are sent all at once, and they will be evaluated with the same garbled input $\tilde{x}$, we can conceptually think of $C^1, C^2, \ldots, C^q$ as disjoint sub-circuits (which have no wires in common) of one big circuit $\mathbb{C}$. Let us define $\mathbb{C} = [C^1, C^2, \ldots, C^q]$.

Next, we observe that the garbling function $\mathsf{NGCircuit}^\star$ is such that garbling circuits $(C^1, C^2, \ldots, C^q)$ one at the time will generate a garbled circuit which is equivalent to the one obtained by garbling $\mathbb{C}$ as a single circuit. To see why, note that the garbling function $\mathsf{NGCircuit}^\star$ operates by encrypting one gate at the time, and only connected gates have correlated keys. As $(C^1, C^2, \ldots, C^q)$ are disjoint, they are encrypted separately regardless of whether they are presented as a single circuit $\mathbb{C}$ or as many independent circuits. Therefore, we can group the view of adversary as follows:

$$\begin{array}{lcl} \tilde{\mathbb{C}}_L & = & (\mathsf{C_L}^1, \ldots, \mathsf{C_L}^q) \\ \tilde{\mathbb{C}}_R & = & (\mathsf{C_R}^1, \ldots, \mathsf{C_R}^q) \\ \mathbb{S}_L & = & (\mathsf{SG}^1, \ldots, \mathsf{SG}^q) \\ \tilde{x} & = & (K_L, \mathsf{key}_L, K_R, \mathsf{key}_R) \end{array}$$

Finally, recall that the flag active is set once and for all in $\mathsf{FE.Setup}$ (Fig. 8) That is, either $L = \mathsf{active}$ and $R = \mathsf{passive}$, or viceversa. Therefore, we can further represent the view of the adversary as follows:

$\tilde{\mathbb{C}}_{\mathsf{active}}, \tilde{x}_{\mathsf{active}}$
$\tilde{\mathbb{C}}_{\mathsf{passive}}, \tilde{x}_{\mathsf{passive}}$
$\mathbb{S}$

This view fits the template of high-level hybrids shown in Figure 6. The exact same arguments then follow to show that GAME_0 and GAME_1 are indistinguishable. In GAME_b , $\tilde{x}_{\mathsf{active}}$ and , $\tilde{x}_{\mathsf{passive}}$ are both garbling of m_b.

Following the same template, the proof strategy is to move from GAME_0, where $\tilde{x}_{\mathsf{active}}$ and $\tilde{x}_{\mathsf{passive}}$ are garbling of m_0, to intermediate games where $\tilde{x}_{\mathsf{passive}}$ is a garbling of m_1 and finally change $\tilde{x}_{\mathsf{active}}$ into garbling of m_1 and thus reaching GAME_1.

Theorem 2. *Assuming the existence of one-way functions, Π is a many functions single message adaptive secure private-key functional encryption, for all circuits in $\mathcal{C}[\mathsf{poly}(\lambda), t]$, with ciphertext size $(n+t)\mathsf{poly}(\lambda)$, where n is the length of the plaintext.*

Proof. It follows from the proof of Theorem 2 applied to the circuit $\mathbb{C}$ defined above. □

7.4 Extensions

We leave as an extension to consider a *full adaptive* security definition for functional encryption where the adversary can choose the functional queries adaptively [ABSV15]. Concretely, this means that the adversary can choose functions adaptively based on the garbled circuits received so far.

To prove security of our construction in this setting, one needs to prove that the underlying garbling scheme $(\mathsf{NGCircuit}^\star, \mathsf{NGInput}, \mathsf{NEval})$ satisfies a stronger adaptivity property that we call *many-time* adaptive security. That is, in the security experiment the adversary is allowed to adaptively ask for many garbled circuits and then choose an single input to evaluate all of them.

Showing that $(\mathsf{NGCircuit}^\star, \mathsf{NGInput}, \mathsf{NEval})$ achieves this stronger property amounts to show that the underlying new somewhere-equivocal encryption scheme (Definition B) achieves a stronger security property where the adversary can choose the plaintexts adaptively on the ciphertexts received so far.

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 657–677, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 166–184, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan

Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.

[JKK+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. Lecture Notes in Computer Science, pages 133–163, Santa Barbara, CA, USA, 2017. Springer, Heidelberg, Germany.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 433–458, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Symmetric-Key Encryption with Special Correctness [LP09]

In our construction of the garbling scheme, we use a symmetric-key encryption scheme $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which satisfies the standard definition of CPA security and an additional *special correctness* property below (this is a simplified and sufficient variant of the property described in from [LP09]). We need this property to ensure the correctness of our garbled circuit construction.

Definition 4 (Special Correctness)**.** A CPA-secure symmetric-key encryption $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies special correctness if there is some negligible function ε such that for any message m we have:

$$\Pr[\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m)) \neq \perp \ : \ k_1, k_2 \leftarrow \mathsf{Gen}(1^\lambda)] \leq \varepsilon(\lambda).$$

Construction. Let $F = \{f_k\}$ be a family of pseudorandom functions where $f_k : \{0,1\}^\lambda \to \{0,1\}^{\lambda+s}$, for $k \in \{0,1\}^\lambda$ and s is a parameter denoting the message length. Define $\mathsf{Enc}_k(m) = (r, f_k(r) \oplus m0^\lambda)$ where $m \in \{0,1\}^s$, $r \xleftarrow{\$} \{0,1\}^\lambda$ and $m0^\lambda$ denotes the concatenation of m with a string of 0s of length λ. Define $\mathsf{Dec}_k(c)$ which parses $c = (r, z)$, computes $w = z \oplus f_k(r)$ and if the last λ bits of w are 0's it outputs the first s bits of w, else it outputs $\perp$.

It's easy to see that this scheme is CPA secure and that it satisfies the special correctness property.

Double Encryption Encryption Security. For convenience, we define a notion of double encryption security, following [LP09]. This notion is implied by standard CPA security but is more convenient to use in our security proof of garbled circuit security.

Definition 5 (Double-encryption security)**.** An encryption scheme $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

- is $(T(\lambda), \varepsilon(\lambda))$-secure under *chosen double encryption* if

$$\mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{\mathsf{double}}(\lambda, 0), \mathsf{Exp}^{\mathsf{double}}(\lambda, 1)\right] = \varepsilon(\lambda).$$

- is secure under *chosen double encryption* if

$$\mathsf{Exp}^{\mathsf{double}}(\lambda, 0) \overset{\text{comp}}{\approx} \mathsf{Exp}^{\mathsf{double}}(\lambda, 1).$$

- is sub-exponentially secure if

$$\exists\ \nu > 0, \forall\ T(\lambda) \in \mathsf{poly}(\lambda)\ \ \mathbf{D}_{T(\lambda)}\left[\mathsf{Exp}^{\mathsf{double}}(\lambda, 1), \mathsf{Exp}^{\mathsf{double}}(\lambda, 0)\right] \leq \varepsilon(\lambda) = 1/2^{\lambda^{\nu}}.$$

where the experiment $\mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}$ is defined as follows.

Experiment $\mathsf{Exp}^{\mathsf{double}}_{\mathcal{A}}(\lambda, b)$

1. The adversary $\mathcal{A}$ on input 1^{λ} outputs two keys k_a and k_b of length λ and two triples of messages (x_0, y_0, z_0) and (x_1, y_1, z_1) where all messages are of the same length.
2. Two keys $k'_a, k'_b \overset{\$}{\leftarrow} \mathsf{Gen}(1^{\lambda})$ are chosen.
3. $\mathcal{A}^{\mathsf{Enc}_{k'_a}(\cdot), \mathsf{Enc}_{k'_b}(\cdot)}$ is given the challenge ciphertexts $c_x \leftarrow \mathsf{Enc}_{k_a}(\mathsf{Enc}_{k'_b}(x_b))$, $c_y \leftarrow \mathsf{Enc}_{k'_a}(\mathsf{Enc}_{k_b}(y_b))$, $c_z \leftarrow \mathsf{Enc}_{k'_a}(\mathsf{Enc}_{k'_b}(z_b))$ as well as **oracle access** to $\mathsf{Enc}_{k'_a}(\cdot)$ and $\mathsf{Enc}_{k'_b}(\cdot)$.
4. $\mathcal{A}$ outputs b' which is the output of the experiment.

The following lemma is essentially immediate - see [LP09] for a formal proof.

Lemma 5. If $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is CPA-secure then it is secure under chosen double encryption with the same security parameter.

B Somewhere Equivocal Symmetric-Key Encryption [HJO+16]

Definition 6. A somewhere equivocal encryption scheme with *block-length* s, *message-length* n (in blocks), and *equivocation-parameter* t (all polynomials in the security parameter) is a tuple of probabilistic polynomial algorithms $\Pi = (\mathsf{seKeyGen}, \mathsf{seEnc}, \mathsf{seDec}, \mathsf{SimEnc}, \mathsf{SimKey})$ such that:

- The key generation algorithm $\mathsf{seKeyGen}$ takes as input the security parameter 1^{λ} and outputs a key: $\mathsf{key} \leftarrow \mathsf{seKeyGen}(1^{\lambda})$.
- The encryption algorithm seEnc takes as input a vector of n messages $\overline{m} = m_1, \ldots, m_n$, with $m_i \in \{0,1\}^s$, and a key key, and outputs ciphertext $\overline{c} \leftarrow \mathsf{seEnc}(\mathsf{key}, \overline{m})$.
- The decryption algorithm seDec takes as input ciphertext $\overline{c}$ and a key key and outputs a vector of messages $\overline{m} = m_1, \ldots, m_n$. Namely, $\overline{m} \leftarrow \mathsf{seDec}(\mathsf{key}, \overline{c})$.
- The simulated encryption algorithm SimEnc takes as input a set of indexes $I \subset [n]$, such that $|I| \leq t$, and a vector of $n - |I|$ messages $(m_i)_{i \notin I}$ and outputs ciphertext $\overline{c}$, and a state state. Namely, $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$.

- The simulated key algorithm SimKey, takes as input the variable state and messages $(m_i)_{i\in I}$ and outputs a key key'. Namely, $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i\in I})$.

and satisfies the following properties:

Correctness. For every $\mathsf{key} \leftarrow \mathsf{seKeyGen}(1^\lambda)$, for every $\overline{m} \in \{0,1\}^{s\times n}$ it holds that:

$$\mathsf{seDec}(\mathsf{key}, (\mathsf{seEnc}(\mathsf{key}, \overline{m})) = \overline{m}$$

Simulation with No Holes. We require that the distribution of $(\overline{c}, \mathsf{key})$ computed via $(\overline{c}, \mathsf{state}) \leftarrow \mathsf{SimEnc}(\overline{m}, \emptyset)$ and $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, \emptyset)$ to be identical to $\mathsf{key} \leftarrow \mathsf{seKeyGen}(1^\lambda)$ and $\overline{c} \leftarrow \mathsf{seEnc}(\mathsf{key}, \overline{m})$. In other words, simulation when there are no holes (i.e., $I = \emptyset$) is identical to honest key generation and encryption.

Security. For any PPT adversary $\mathcal{A}$, there exists a negligible function $\nu = \nu(\lambda)$ such that:

$$\Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 1) = 1] \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$ is defined as follows:

Experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, b)$

1. The adversary $\mathcal{A}$ on input 1^λ outputs a set $I \subseteq [n]$ s.t. $|I| < t$, vector $(m_i)_{i\notin I}$, and a challenge index $j \in [n] \setminus I$. Let $I' = I \cup j$.
2. – If $b = 0$, compute $\overline{c}$ as follows: $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i\notin I}, I)$.
   – If $b = 1$, compute $\overline{c}$ as follows: $(\mathsf{state}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i\notin I'}, I')$.
3. Send $\overline{c}$ to the adversary $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs the set of remaining messages $(m_i)_{i\in I}$.
   – If $b = 0$, compute key as follows: $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i\in I})$.
   – If $b = 1$, compute key as follows: $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{state}, (m_i)_{i\in I'})$.
5. Send key to the adversary $\mathcal{A}$.
6. $\mathcal{A}$ outputs b' which is the output of the experiment.

In [HJO+16], a somewhere equivocal encryption is constructed from one-way functions, proving the following theorem.

Theorem 3. *Assuming the existence of one-way functions, there exists a somewhere equivocal encryption scheme for any polynomial message-length n, block-length s, and equivocation parameter t, having key size $t \cdot s \cdot \mathsf{poly}(\lambda)$ and ciphertext of size $n \cdot s$ bits.*

Extension. Such construction naturally extends to a modified encryption algorithm $\mathsf{seEnc}^\star$, that instead of taking in input the entire vector $\overline{m} = m_1, \ldots, m_n$, it takes in input a few blocks that arrive in a streaming fashion. Namely, $\mathsf{seEnc}^\star$ takes as input an upperbound N, a vector of $j \geq 1$ messages $\overline{m} = m_1, \ldots, m_j$, and a key key and it outputs j encryptions, while keeping a counter on the number of encryptions computed so far. The messages are encrypted as long as the counter is less than the upper bound N.

To see why the implementation provided in [HJO+16]also supports the modified version $\mathsf{seEnc}^\star$, note that their encryption is performed by xoring the output of a special pseudo-random function (PRF) with the plaintext. To encrypt n blocks, one evaluates the PRF on inputs $1, 2, \ldots, n$ and then xor the result with the blocks. Naturally, one can encrypt any number of blocks at different times. The construction will still work provided that the algorithm is stateful and remembers the last index on which the PRF has been evaluated on (so that the same PRF evaluation is not used twice).

Concering security, for our application it suffices that $\mathsf{seEnc}^\star$ satisfies the same "non-adaptive" definition of security as in experiment $\mathsf{Exp}^{\mathsf{simenc}}$ where the adversary needs to commit to the *entire* vector $(m_i)_{i\notin I}$ in advance.

C Hybrid Games of [HJO+16]

Gate/Circuit Configuration. We start by defining a *gate configuration*. A gate configuration is a pair (outer mode, garbling mode) indicating the way a gate is computed. The outer encryption mode can be $\{\mathsf{EquivEnc}, \mathsf{BindEnc}\}$ depending on whether the outer encryption contains a "hole" in place of that gate or whether it is binding on that gate. The garbling mode can be $\{\mathsf{RealGate}, \mathsf{SimGate}, \mathsf{InputDepSimGate}\}$ which corresponds to the distributions outlined in Figure 9. We stress that, if the garbling mode of a gate is InputDepSimGate then we require that the outer encryption mode is EquivEnc. This means that there are 5 valid gate configurations for each gate.

RealGate
$c_{0,0} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{g(0,0)}))$
$c_{0,1} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{g(0,1)}))$
$c_{1,0} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{g(1,0)}))$
$c_{1,1} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{g(1,1)}))$

SimGate
$c_{0,0} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^0))$
$c_{0,1} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^0))$
$c_{1,0} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^0))$
$c_{1,1} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^0))$

InputDepSimGate
$c_{0,0} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)}))$
$c_{0,1} \leftarrow \mathsf{Enc}_{k_a^0}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)}))$
$c_{1,0} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^0}(k_c^{v(c)}))$
$c_{1,1} \leftarrow \mathsf{Enc}_{k_a^1}(\mathsf{Enc}_{k_b^1}(k_c^{v(c)}))$

Figure 8: Garbling Gate modes: RealGate (left), SimGate (center), InputDepSimGate (right). The value $v(c)$ depends on the input x and corresponds to the bit going over the wire c in the computation $C(x)$.

A *circuit configuration* simply consists of the gate configuration for each gate in the circuit. More specifically, we represent a circuit configuration by a tuple $(I, (\mathsf{mode}_i)_{i\in[q]})$ where

- Set $I \subseteq [q]$ contains the indices of the gates i whose outer mode is EquivEnc.
- The value $\mathsf{mode}_i \in \{\mathsf{RealGate}, \mathsf{SimGate}, \mathsf{InputDepSimGate}\}$ describes the garbling mode of gate i.

A *valid circuit configuration* is one where all indexes i such that $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ satisfy $i \in I$.

Game $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$

Garble Circuit C:

– Garble Gates

(Wires) $k_{w_i}^\sigma \leftarrow \mathsf{Gen}(1^\lambda)$ for $i \in [p]$, $\sigma \in \{0,1\}$.

(Gates) For each $\mathsf{gate}_i = (g, w_a, w_b, w_c)$ in C.

- If $\mathsf{mode}_i = \mathsf{RealGate}$: run $\widetilde{g}_i \leftarrow \mathsf{GarbleGate}(g, \{k_{w_a}^\sigma, k_{w_b}^\sigma, k_{w_c}^\sigma\}_{\sigma\in\{0,1\}})$.
- if $\mathsf{mode}_i = \mathsf{SimGate}$: run $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}(\{k_{w_a}^\sigma, k_{w_b}^\sigma\}_{\sigma\in\{0,1\}}, k_{w_c}^0)$.

– Outer Encryption.

1. $(\mathsf{state}, \widetilde{C}) \leftarrow \mathsf{SimEnc}((\widetilde{g}_i)_{i\notin I}, I)$.
2. Output $\widetilde{C}$.

Garble Input x:

(Compute adaptive gates)

For each $i \in I$ s.t. $\mathsf{mode}_i = \mathsf{InputDepSimGate}$:

Let $\mathsf{gate}_i = (g_i, w_a, w_b, w_c)$, and let $v(c)$ be the bit on the wire w_c during the computation $C(x)$.

Set $\widetilde{g}_i \leftarrow \mathsf{GarbleSimGate}((k_{w_a}^\sigma, k_{w_b}^\sigma)_{\sigma\in\{0,1\}}, k_{w_c}^{v(c)})$.

(Decryption key) $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{state}, (\widetilde{g}_i)_{i\in I})$

(Output tables) Let $y = C(x)$. For $j = 1, \ldots, m$:

Let i be the index of the gate with output wire out_j.

- If $\mathsf{mode}_i \neq \mathsf{SimGate}$, set $\widetilde{d}_j := [(k_{\mathsf{out}_j}^0 \to 0), (k_{\mathsf{out}_j}^1 \to 1)]$,
- else, set $\widetilde{d}_j := [(k_{\mathsf{out}_j}^{y_j} \to 0), (k_{\mathsf{out}_j}^{1-y_j} \to 1)]$.

(Select input keys) For $j = 1, \ldots, n$:

- If all gates i having in_j as an input wire satisfy $\mathsf{mode}_i = \mathsf{SimGate}$, then set $K[i] := k_{\mathsf{in}_i}^0$,
- else set $K[i] := k_{\mathsf{in}_i}^{x_i}$.

Output $\widetilde{x} := (K, \mathsf{key}', \{\widetilde{d}_j\}_{j\in[m]})$.

Figure 9: The Hybrid Game.

The Hybrid Game $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$. Every valid circuit configuration I, $(\mathsf{mode}_i)_{i\in[q]}$ defines a hybrid game $\mathsf{Hyb}(I, (\mathsf{mode}_i)_{i\in[q]})$ as specified formally Figure 10 and described informally below. The hybrid game consists of two procedures: $\mathsf{GCircuit}'$ for creating the garbled circuit $\widetilde{C}$ and GInput' for creating the garbled input $\tilde{x}$ respectively. The garbled circuit is created by picking random keys $k^{\sigma}_{w_j}$ for each wire w_j. For each gate i, such that $\mathsf{mode}_i \in \{\mathsf{RealGate}, \mathsf{SimGate}\}$ it creates a garbled gate $\widetilde{g}_i$ using the corresponding distribution as described in Figure 9. The garbled circuit $\widetilde{C}$ is then created by simulating the outer encryption using the values $\widetilde{g}_i$ in locations $i \notin I$ and "holes" in the locations I. The garbled input is created by first sampling the garbled gates $\widetilde{g}_i$ for each i such that $\mathsf{mode}_i = \mathsf{InputDepSimGate}$ using the corresponding distribution in Figure 9 and using knowledge of the input x. Then the decryption key key is simulated by plugging in the holes in locations I with the correctly sampled garbled gates $\widetilde{g}_i$. There is some subtlety about how the input labels $K[i]$ and the output label maps $\widetilde{d}_j$ are created when computing $\tilde{x}$:

- If all of the gates having in_i as an input wire are in $\mathsf{SimGate}$ mode, then $K[i] := k^0_{\mathsf{in}_i}$ else $K[i] := k^{x_i}_{\mathsf{in}_i}$.
- If the unique gate having out_j as an output wire is in $\mathsf{SimGate}$ mode, then we give the simulated output map $\widetilde{d}_j := [(k^{y_j}_{\mathsf{out}_j} \to 0), (k^{1-y_j}_{\mathsf{out}_j} \to 1)]$ else the real one $\widetilde{d}_j := [(k^0_{\mathsf{out}_j} \to 0), (k^1_{\mathsf{out}_j} \to 1)]$.

Real game and Simulated Game. By definition of adaptively secure garbled circuits (Definition 2), the real game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 0)$ is equivalent to $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i\in[q]})$ and the simulated game $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 1)$ is equivalent to $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i\in[q]})$. Therefore, the main aim is to show that these hybrids are indistinguishable.[3]

C.1 Rules for Indistinguishable Hybrids

Next, we provide rules that allow us to move from one configuration to another and prove that the corresponding hybrid games are indistinguishable. We define three rules that allow us to do this. We define $\mathsf{mode} \stackrel{\text{def}}{=} (\mathsf{mode}_i)_{i\in[q]}$.

C.1.1 Indistinguishability Rule 1: Changing the Outer Encryption Mode $\mathsf{BindEnc} \leftrightarrow \mathsf{EquivEnc}$.

This rule allows to change the outer encryption of a single gate. It says that one can move from a valid circuit configuration (I, mode) to a circuit configuration (I', mode) where $I' = I \cup j$. Thus one more gate is now computed equivocally (and vice versa).

Lemma 6. Let (I, mode) be any valid circuit configuration, let $j \in [q] \setminus I$ and let $I' = I \cup j$. Then $\mathsf{Hyb}(I, \mathsf{mode}) \stackrel{\text{comp}}{\approx} \mathsf{Hyb}(I', \mathsf{mode})$ are computationally indistinguishable as long as $\Pi = (\mathsf{seKeyGen}, \mathsf{seEnc}, \mathsf{seDec}, \mathsf{SimEnc}, \mathsf{SimKey})$ is a somewhere equivocal encryption scheme with equivocation parameter t such that $|I'| \le t$.

Definition 7 (Predecessor/Successor/Sibling Gates [HJO+16])**.** Given a circuit C and a gate $j \in [q]$ of the form $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ with incoming wires w_a, w_b and outgoing wire w_c:

- We define the *predecessors* of j, denoted by $\mathsf{Pred}(j)$, to be the set of gates whose outgoing wires are either w_a or w_b. If w_a, w_b are input wires then $\mathsf{Pred}(j) = \emptyset$, else $|\mathsf{Pred}(j)| = 2$.
- We define the *successors* of j, denoted by $\mathsf{Succ}(j)$ to be the set of gates that contain w_c as an incoming wire. If w_c is an output wires then $\mathsf{Succ}(j) = \emptyset$.
- We define the *siblings* of j, denoted by $\mathsf{Siblings}(j)$ to be the set of gates that contain either w_a or w_b as an incoming wire.

[3]Note that, the games $\mathsf{Hyb}(\cdots)$ use the simulated encryption and key generation procedures of the somewhere equivocal encryption, while the games $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, b)$ only use the real key generation and encryption procedures. However, by definition, these are equivalent when $I = \emptyset$ (no "holes").

C.1.2 Indistinguishability Rule 2. Changing the Garbling Mode $\mathsf{RealGate} \leftrightarrow \mathsf{InputDepSimGate}$

This rule allows us to change the mode of a gate j from $\mathsf{RealGate}$ to $\mathsf{InputDepSimGate}$ as long as $j \in I$ and that $\mathsf{gate}_j = (g, w_a, w_b, w_c)$ has incoming wires w_a, w_b that are either input wires or are the outgoing wires of some predecessor gates both of which are in $\mathsf{InputDepSimGate}$ mode.

Lemma 7. Let $(I, \mathsf{mode} = (\mathsf{mode}_i)_{i\in[q]})$ be a valid circuit configuration and let $j \in I$ be an index such that $\mathsf{mode}_j = \mathsf{RealGate}$ and for all $i \in \mathsf{Pred}(j)$: $\mathsf{mode}_i = \mathsf{InputDepSimGate}$. Let $\mathsf{mode}' = (\mathsf{mode}'_i)_{i\in[q]}$ be defined by $\mathsf{mode}'_i = \mathsf{mode}_i$ for all $i \neq j$ and $\mathsf{mode}'_j = \mathsf{InputDepSimGate}$. Then the games $\mathsf{Hyb}(I, \mathsf{mode}) \overset{\text{comp}}{\approx} \mathsf{Hyb}(I, \mathsf{mode}')$ are computationally indistinguishable as long as $\Gamma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an encryption scheme secure under *chosen double encryption*.

C.1.3 Indistinguishability Rule 3. Changing the Garbling Mode: $\mathsf{InputDepSimGate} \leftrightarrow \mathsf{SimGate}$.

This rule allows us to change the mode of a gate j from $\mathsf{InputDepSimGate}$ to $\mathsf{SimGate}$ under the condition that all successor gates $i \in \mathsf{Succ}(j)$ satisfy that $\mathsf{mode}_i \in \{\mathsf{InputDepSimGate}, \mathsf{SimGate}\}$.

Lemma 8. Let $(I, \mathsf{mode} = (\mathsf{mode}_i)_{i\in[q]})$ be a valid circuit configuration and let $j \in I$ be an index such that $\mathsf{mode}_j = \mathsf{InputDepSimGate}$ and for all $i \in \mathsf{Succ}(j)$ we have $\mathsf{mode}_i \in \{\mathsf{SimGate}, \mathsf{InputDepSimGate}\}$. Let $\mathsf{mode}' = (\mathsf{mode}'_i)_{i\in[q]}$ be defined by $\mathsf{mode}'_i = \mathsf{mode}_i$ for all $i \neq j$ and $\mathsf{mode}'_j = \mathsf{SimGate}$. Then the games $\mathsf{Hyb}(I, \mathsf{mode}) \equiv \mathsf{Hyb}(I, \mathsf{mode}')$ are identically distributed.

C.2 Pebbling and Sequences of Hybrid Games

In the last section we defined hybrid games parameterized by a configuration (I, mode). We also gave 3 rules, which describe ways that allow us to indistinguishably move from one configuration to another. Now our goal is to use the given rules so as to define a *sequence of indistinguishable hybrid games* that takes us from the *real game* $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i\in[q]})$ to the simulation $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i\in[q]})$.

Pebbling Game. We show that the problem of finding such sequences of hybrid games can be captured by a certain type of *pebbling game* on the circuit C. Each gate can either have *no pebble*, a *black pebble*, or *a gray pebble* on it (this will correspond to $\mathsf{RealGate}, \mathsf{InputDepSimGate}$ and $\mathsf{SimGate}$ modes respectively). Initially, the circuit starts out with no pebbles on any gate. The game consist of the following possible moves:

Rule A. We can place or remove a black pebble on a gate as long as both predecessors of that gate have black pebbles (or the gate is an input gate).

Rule B. We can replace a black pebble with a gray one, only if successors of that gate have black or gray pebbles on them (or the gate is an output gate).

A *pebbling* of a circuit C is a sequence of γ moves that follow rules A and B and that end up with a gray pebble on every gate. We say that a pebbling uses t black pebbles if this is the maximal number of black pebbles on the circuit at any point in time during the game.

From Pebbling to Sequence of Hybrids. In next theorem we prove that any pebbling of a circuit C results in a sequence of hybrids that shows indistinguishability of the real and simulated games. The number of hybrids is proportional to the number of moves in the pebbling and the equivocation parameter is proportional to the number of black pebbles it uses.

Theorem 4. *Assume that there is a pebbling of the circuit C in γ moves. Then there is a sequence of $2 \cdot \gamma + 1$ hybrid games, starting with the real game $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{RealGate})_{i\in[q]})$ and ending with the simulated game $\mathsf{Hyb}(I = \emptyset, (\mathsf{mode}_i = \mathsf{SimGate})_{i\in[q]})$ such that any two adjacent hybrid games in the sequence are indistinguishable by rules 1,2 or 3 from the previous section. Furthermore if pebbling uses t^* black pebbles then every hybrid $\mathsf{Hyb}(I, \mathsf{mode})$ in the sequence satisfies $|I| \leq t^*$. In particular, indistinguishability holds as long as the equivocation parameter is at least t^*.*

D Pebbling Strategies [HJO+16]

In this section we give two pebbling strategies for arbitrary circuit with width w, depth d, and q gates. The first strategy uses $O(q)$ moves and $O(w)$ black pebbles. The second strategy uses $O(q2^d)$ moves and $O(d)$ black pebbles.

D.0.1 Strategy 1

To pebble the circuit proceed as follows:

$\mathsf{Pebble}(C)$:

1. Put a black pebble on each gate at the input level (level 1).
2. For $i = 1$ to $d - 1$, repeat:
   (a) Put a black pebble on each gate at level $i + 1$.
   (b) For each gate at level i, replace the black pebble with a gray pebble.
   (c) $i \leftarrow i + 1$
3. For each gate at level d, replace the black pebble with a gray pebble.

This strategy uses $\gamma = 2q$ moves and $t^* = 2w$ black pebbles.

D.0.2 Strategy 2

This is a recursive strategy defined as follows.

- $\mathsf{Pebble}(C)$:
  For each gate i in C starting with the gates at the top level moving to the bottom level:
  1. $\mathsf{RecPutBlack}(C, i)$
  2. Replace the black pebble on gate i with a gray pebble.
- $\mathsf{RecPutBlack}(C, i)$: // Let $\mathsf{LeftPred}(C, i)$ and $\mathsf{RightPred}(C, i)$ be the two predecessors of gate i in C.
  1. If gate i is an input gate, put a black pebble on i and **return**.
  2. Run $\mathsf{RecPutBlack}(C, \mathsf{LeftPred}(C, i))$, $\mathsf{RecPutBlack}(C, \mathsf{RightPred}(C, i))$
  3. Put a black pebble on gate i.
  4. Run $\mathsf{RecRemoveBlack}(C, \mathsf{LeftPred}(C, i))$ and $\mathsf{RecRemoveBlack}(C, \mathsf{RightPred}(C, i))$,
- $\mathsf{RecRemoveBlack}(C, i)$: This is the same as $\mathsf{RecPutBlack}$, except that instead of putting a black pebble on gate i, in steps 1 and 3, we remove it.

The above gives us a strategy to pebble any circuit with at most $\gamma = q4^d$ moves and $t = 2d$ black pebbles.